Implementing Molecular Dynamics on Hybrid High Performance Computers - Three-Body Potentials

W. Michael Brown[a,*], Masako Yamada[b]

[a]National Center for Computational Sciences, Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA
[b]GE Global Research, 1 Research Circle K1-3A17A, Nisakyuna, New York, USA

Abstract

The use of coprocessors or accelerators such as graphics processing units (GPUs) has become popular in scientific computing applications due to their low cost, impressive floating-point capabilities, high memory bandwidth, and low electrical power requirements. Hybrid high-performance computers, defined as machines with nodes containing more than one type of floating-point processor (e.g. CPU and GPU), are now becoming more prevalent due to these advantages. Although there has been extensive research into methods to use accelerators efficiently to improve the performance of molecular dynamics (MD) codes employing pairwise potential energy models, little is reported in the literature for models that include many-body effects. 3-body terms are required for many popular potentials such as MEAM, Tersoff, REBO, AIREBO, Stillinger-Weber, Bond-Order Potentials, and others. Because the per-atom simulation times are much higher for models incorporating 3-body terms, there is a clear need for efficient algorithms usable on hybrid high performance computers. Here, we report a shared-memory force-decomposition for 3-body potentials that avoids memory conflicts to allow for a deterministic code with substantial performance improvements on hybrid machines. We describe modifications necessary for use in distributed memory MD codes and show results for the simulation of water with Stillinger-Weber on the hybrid Titan supercomputer. We compare performance of the 3-body model to the SPC/E water model when using accelerators. Finally, we demonstrate that our approach can attain a speedup of 5.1 with acceleration on Titan for production simulations to study water droplet freezing on a surface.

Keywords:
Molecular dynamics, 3-body, GPU, coprocessor, accelerator, Stillinger-Weber

1. Introduction

Issues with power consumption, heat dissipation, and high memory access latencies have made heterogeneous architectures a popular idea for increasing parallelism with electrical power and cost efficiency. Basic heterogeneous architectures include hybrid systems that combine a traditional CPU with a coprocessor or accelerator such as a graphics processing unit (GPU), digital signal processor, field-programmable gate array, or other many-core chip. These architectures are becoming more popular in high-performance computers due to significant advantages in the performance to electrical power ratio; for example, the upgrade from the CPU-only Jaguar Cray XT5 at Oak Ridge National Laboratory to the hybrid Titan Cray XK7 resulted in ten times the observed performance while requiring only 19% more electrical power [1]. Not only was Titan the fastest ranked supercomputer at the time of writing, it was also ranked number 3 in terms of power efficiency[2, 3].

In order to make effective use of hybrid machines, changes to the models, algorithms, and/or code are typically required. For the latter, changes are often required in order to 1) efficiently use shared memory parallelism, 2) increase concurrency with fine-grain parallelism, and 3) improve data locality, often with explicit code to improve hierarchical memory use. There has been extensive research along these lines to demonstrate significant performance improvements for molecular dynamics on hybrid machines [4, 5]. Most of this work has been focused on pairwise potentials. Although these potentials are commonly employed in the simulation of polymers and biomolecules, many materials such as metals, covalent solids, and carbon nanotubes, as well as chemical reactions, are typically simulated with potential energy models that incorporate many-body effects. These potentials typically have a much higher computational cost per atom when compared to pairwise potentials. The simulation of materials with many-body potentials has been described in the context of the "Law of Constancy of Pain" [6] - the trend in the development of new many-body potentials has been to use increased CPU speeds and core counts not for faster simulations, but to simulate with more complex models that have improved accuracy and transferability.

For these reasons, it is clearly desirable to develop algorithms and code for simulation of many-body potentials on accelerators [6, 7]. Despite their importance, very little has been reported in the literature describing methods or performance gains from acceleration of many-body potentials. In part, this

*Corresponding author.
Email addresses: brownw@ornl.gov (W. Michael Brown), yamada@ge.com (Masako Yamada)

could be due to the increased complexity of these models - the models require multiple and/or nested loops that increase data dependencies, require changes to the standard neighbor list used in pairwise models, and can require additional communications in parallel codes [6]. Implementations of the embedded atom method (EAM) [8], for use on accelerators and coprocessors have been described that led to significant performance improvements [9, 10]. The EAM potential incorporates the energy from embedding an atom into the electron density produced by its neighbors. In this sense, the EAM potential is many-body because the electron charge density at each neighboring atom position must be calculated with a loop over surrounding atoms within some cutoff. However, this model is somewhat unique among many-body potentials in that it can still be computed using only pairwise summations. While this requires additional interprocess communications during the force computation, we have shown that parallel implementations on hybrid machines can maintain significant performance improvements up to the entire 900 nodes available at the time of study[10].

For other many-body potentials, the data dependencies are more complex. 3-body interactions are commonly used and require terms calculated for every triplet of atoms in addition to every pair. 3-body terms are required for many popular potentials such as MEAM [11], Tersoff [12], REBO [13], AIREBO [14], Stillinger-Weber [15], Bond-Order Potentials [16], and others. Although the nested loops required for 3-body terms are simple to implement for serial calculations, their implementation for many-core accelerators/coprocessors results in some complications. The problems arise because non-uniform memory access and limited per-core memory typically favor shared-memory atom- or force-decomposition for parallelism. The naïve implementations of these decompositions result in data dependencies; the evaluation of each energy term in the summation is used to update the force of three different atoms. Therefore, naïve implementations require the use of atomic operations to prevent memory collisions - erroneous results caused by simultaneous update of the same location in memory from multiple threads. Atomic operations are generally undesirable because of the high latencies and because they introduce randomness into the code. Many experienced developers will prefer deterministic code whenever possible to make debugging feasible on high performance computers that are constantly changing out hardware and often changing the software stack.

Alternative implementations can alter the force computation loops to avoid data dependencies in exchange for increased computation. These implementations can reduce global memory access and allow for deterministic code, but the potential performance gains become limited due to substantial increases in the amount of floating point operations required. Although an elegant approach for implementing the Stillinger-Weber potential on GPUs has been described with impressive performance [17], the approach is only applicable for simulations of solid crystals where the neighbors of any given atom do not change. Therefore, the approach is not applicable to many problems such as vacancy diffusion or the simulation of liquids.

In this paper, we present a simple approach for computing 3-body interactions using an atom or force decomposition in shared memory. The approach avoids data dependencies allowing for a deterministic code. We present the changes necessary for implementation in parallel molecular dynamics codes using a spatial decomposition. We provide benchmark results on a hybrid Cray XK7 supercomputer for a 3-body implementation building on our previous work in the LAMMPS molecular dynamics package[18, 19, 10]. We evaluate performance using the mW water model. The mW water model is comprised of a single effective particle that preferentially forms four tetrahedral bonds. The model has no explicit charges, and hence no hydrogen-bonds or long-range electrostatic terms, but it reproduces the quantitative behavior of water as well as or better than conventional 3, 4 or 5 point charge models. Simulation rates have been reported that are 180 times faster than the least expensive 3 point charge model (specifically SPC/E) while the quantitative agreement of the melting temperature, enthalpy of melting, liquid-vacuum surface tension and liquid density as a function of temperature have been shown to be superior to the SPC, SPC/E, TIP3P, TIP4P and TIP5P models [20]. The orders-of-magnitude speedup relative to SPC/E has been attributed to: a) a three-fold reduction in number of atoms, b) the elimination of expensive k-space solvers and c) the enabling of longer time steps due to the lack of internal bonds. In particular, the mW model has been shown to facilitate the observation of spontaneous freezing in water [21] with much fewer timesteps relative to well-known traditional point-charge potentials [22, 23] while still reproducing many quantitative water properties of interest. This makes molecular dynamics a more attractive tool to probe phenomena that span many orders of magnitude of space and/or time, such as our particular area of interest, which is the study of ice formation in the presence of surfaces.

Here, we evaluate performance of the mW model with acceleration compared to both the standard CPU implementation for Stillinger-Weber in LAMMPS and simulation with the SPC/E water model. Our benchmark simulations include periodic water boxes and production simulations that are used to study the microscopic mechanism of droplet freezing on a surface. For the latter, simulation sizes of one million water molecules are used in order to probe the types of complex crystallization behaviors [24] we have observed experimentally for water droplets freezing on surfaces [25, 26].

2. Methods

2.1. LAMMPS

Our implementation for 3-body potentials has been performed within the LAMMPS molecular dynamics package [27]. LAMMPS is parallelized via MPI, using spatial-decomposition techniques that partition the 3D simulation domain into a grid of smaller 3D subdomains, one per MPI process. The algorithms we have previously developed for pairwise potentials and long-range electrostatics on accelerators/-coprocessors supporting CUDA or OpenCL in LAMMPS have been published in detail [18, 19, 10]. LAMMPS supports acceleration for short-range force calculation [18] with optional acceleration for neighbor list builds and/or (P^3M) long-range

electrostatics [19]. Neighbor list builds are performed on the accelerator by first constructing a cell list that is utilized to build a Verlet list using a radix sort to assert deterministic results. The van der Waals and short-range electrostatic forces are computed in a separate kernel. For each particle, the force-accumulation is performed by one or multiple threads. A default number of threads is chosen based on the hardware and the potential model being used for calculation. The short-range calculation can be performed in single, mixed, or double precision. For mixed precision, all accumulation is performed in double precision and forces, torques, energies, and virials are stored in double precision. For long-range electrostatics, acceleration for P^3M is supported for charge assignment to the mesh and force interpolation. The parallel FFT is performed on the host (see below). The P^3M calculation can be performed in single or double precision.

All of the statistics computations, thermostats, barostats, time integration, bond/angle/dihedral/improper calculations, and any other simulation modifications are performed on the host. In order to achieve efficient acceleration, these calculations must be parallelized within each node on the host [18]. This is performed by using multiple MPI processes, each sharing one or more accelerators on a compute node. This approach has several advantages. Those relevant to the work here include full compatibility with all of the other routines in LAMMPS that run on the CPU, the ability to overlap data transfers and computation from different MPI processes sharing the accelerator, concurrent calculation of non-bonded forces on the accelerator and bonded forces on the host, and concurrent execution of long-range and short-range forces using separate partitions of MPI processes [10]. The downside of the approach is the requirement to determine an optimal number of MPI processes to share the accelerator. This will not necessarily be all cores available on the node for smaller problem sizes due to the overhead for sharing an accelerator on current hardware.

2.2. Accelerator Model

For this work, we consider accelerators and coprocessors that fit a model suited for OpenCL and CUDA. Because OpenCL and CUDA use different terminology, we have listed equivalent (in the context of this paper) terms in Table 1. Here, we will use OpenCL terminology. The *host* consists of CPU cores and associated addressable memory. The *device* is an accelerator consisting of 1 or more *compute units* that typically correspond to processors or multiprocessors in the hardware (note that for OpenCL this device might be the CPU). Each compute unit has multiple *processing elements* that typically correspond to cores in the processor. The device has *global memory* that may or may not be addressable by the CPU, but is shared among all compute units. Additionally, the device has *local memory* for each compute unit that is shared by the processing elements on the compute unit. Each processing element on the device executes instructions from a work-item (this concept is similar to a thread running on a CPU core). We assume that the compute unit might require SIMD instructions in hardware; therefore, branches that could result in divergence of the execution path for different work-items are a concern. In this paper, the problem is referred to as *work-item divergence*. We also assume that global memory latencies can be orders of magnitude higher when compared to local memory access.

Table 1: Equivalent OpenCL and CUDA terminology.

OpenCL	CUDA
Compute Unit	Multiprocessor
Processing Element	Core
Local memory	Shared memory
Work-item	Thread
Work-group	Thread Block
Command Queue	Stream

We assume that access latencies for coalesced memory will be much smaller. Coalesced memory access refers to sequential memory access for data that is correctly aligned in memory. This will happen, for example, when data needed by individual processing elements on a compute unit can be "coalesced" into a larger sequential memory access given an appropriate byte alignment for the data. Consider a case where each processing element needs to access one element in the first row of a matrix with arbitrary size. If the matrix is row-major in memory, the accelerator can potentially use coalesced memory access; if the matrix is column-major, it cannot. The penalties for incorrect alignment or access of non-contiguous memory needed by processing elements will vary depending on the hardware.

A *kernel* is a routine compiled for execution on the device. The work for a kernel is decomposed into a specified number of *work-groups* each with a specified number of *work-items*. Each work-group executes on only one compute unit. The number of work-items in a work-group can exceed the number of physical processing elements on the compute unit, allowing more work-items to share local memory and the potential to hide memory access latencies. The number of registers available per work-item is limited. A device is associated with one or more *command queues*. A command queue stores a set of kernel calls and/or host-device memory transfers that can be executed asynchronously with host code.

2.3. Geryon Library

For our LAMMPS implementation, we have used the Geryon library that provides a succinct API allowing a single code to compile with both CUDA and OpenCL [18]. Currently, OpenCL libraries or beta libraries are available for all major vendors for CPUs, GPUs, accelerators, and coprocessors. The Geryon library is available under the Free-BSD license from http://users.nccs.gov/˜wb8/geryon/index.htm.

2.4. Accelerating 3-Body Interactions

As the name implies, 3-body contributions, U_3, to a potential energy model are evaluated using triplets of atoms instead of pairs,

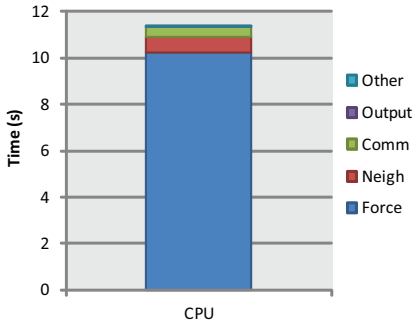

Figure 1: Time for simulation of a 32000 molecule water box with mW on a single CPU using 16 cores. The profile shows time for calculation of forces, energies, and virials (Force), neighbor-list builds (Neigh), MPI communications (Comm), screen/file (Output), and time integration, statistics and other calculations (Other).

$$U_3 = \begin{cases} \sum_i \sum_{j \neq i} \sum_{k>j} \phi(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k) & r_{ij} < r_c, r_{ik} < r_c \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

for atom position $\mathbf{p}_.$ and interatomic separation $r_{..}$. r_c is set to enforce a spherical cutoff to allow for implementations with $O(N)$ time complexity as opposed to $O(N^3)$. Pairwise potentials typically only require half of the atoms within the cutoff to be stored in a neighbor list such that each pair of atoms is only evaluated once during the force loop. Evaluation of equation 1 however, requires neighbor lists storing all atoms within the cutoff. The computational time for simulations employing 3-body interactions is typically almost entirely dominated by the force calculation loop. Figure 1 shows the profile for simulation of a 32,000 atom water box with the 3-body Stillinger-Weber potential. In this case the force calculation is 90% of the entire simulation on a single node and 95% of this time is spent on the 3-body interaction. For some 3-body potentials it is substantially higher. This type of profile is an ideal case for porting to hybrid machines because of the high upper bound to performance gains from running a single routine (or small set of routines) on the accelerator. The observed performance gains, of course, will depend on how well the routines run on the accelerator - how much fine-grain parallelism can be exposed for doing as much computation with as little global memory access as possible.

Although we do use a spatial decomposition for some routines on the device such as kernels for neighbor lists [18], the need for efficient non-uniform memory access coupled with limited per-core memory typically favors shared-memory atom- or force-decomposition for parallelism of the force computation loop. In an atom-decomposition, each work-item iterates through the force loop for a single atom. In a force-decomposition, the terms in the force loop are split between multiple work-items. In either case, the force is computed as the gradient of the potential energy with respect to atom position. For the naïve approach (NA), this means updating the force on three different atoms for each term calculated for the summation in equation 1 in order to minimize the computation:

```
for (i=0; i<n; i++)
  num_nbors=get_neighbor_count(i);
  for (jj=0; jj<num_nbors; jj++) {
    j=neighbor(i,jj);
    if (distance(i,j) >= cutoff) continue;
    for (kk=jj+1; kk<num_nbors; kk++) {
      k=neighbor(i,kk);
      if (distance(i,k) >= cutoff) continue;
      threebody(i,j,k,fi,fj,fk);
      force[i] += fi;
      force[j] += fj;
      force[k] += fk;
    }
  }
```

The NA has several issues that arise when parallelizing for shared memory: (1) parallelization of the jj and kk loops under the $kk > jj$ condition results in more computation for work-items with lower rank, (2) multiple work-items can potentially update the same force location in memory and therefore atomic operations are required, (3) because molecular dynamics codes typically employ methods to sort atoms in memory based on location (to improve data locality), the number of memory collisions encountered during atomic operations will increase resulting in much higher memory access times, and (4) the number of global memory updates required is high. Issue (1) can be addressed with transformations of the 3D matrix of indices for force computations to balance the amount of work between threads [28]. Issue (3) can be addressed with a simple reindexing of the i atom based on the work-item rank [19]. Issues (2) and (4) are more difficult, however. The force updates in the listing above can be modified to use registers such that there is only a single global memory access for each iteration of each of the i, jj, and kk loops. In this case, the number of forces that must be updated in global memory is $n + n \cdot \frac{b_n^2 + b_n}{2}$ for n atoms with b_n neighbors per atom. The global memory access problem is made worse by the requirement to use atomic operations to store the j and k atom forces. This also has the undesirable effect of introducing randomness into the code.

Although the NA minimizes computation, memory access is a much more common bottleneck for force calculations on accelerators and many-core chips. Therefore, approaches that reduce memory access in trade for increased computational requirements can improve performance. For pairwise potentials, this is commonly addressed by doubling the number of force computations to eliminate the possibility of memory collisions. For 3-body interactions, a similar approach can be used; however, it requires more substantial modifications. We refer to this approach as the redundant computation approach (RCA). For the RCA, the force computation is parallelized such that the force on each atom in a given triplet can be computed by different work-items, each performing some redundant computations:

```
for (i=0; i<n; i++)
  num_nbors=get_neighbor_count(i);
  for (jj=0; jj<num_nbors; jj++) {
```

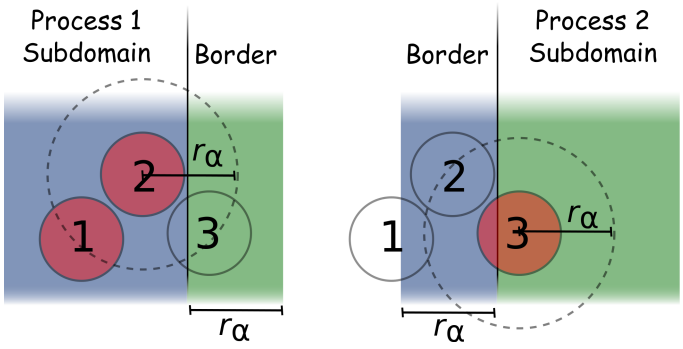

Figure 2: Illustration of a triplet of atoms across the border of two MPI processes. Filled (red) circles denote atoms local to a process; circles without fill are ghost atoms. For the NA, a single neighbor list is used for every triplet and interprocess communication is required for force accumulation. For the RCA, the 3-body interaction loop is calculated over neighbors of neighbors. This requires calculation of neighbor lists for ghost atoms and for the length of the border to be doubled to $2 \cdot r_\alpha$.

```
    j=neighbor(i,jj);
    if (distance(i,j) >= cutoff) continue;

    for (kk=jj+1; kk<num_nbors; kk++) {
      k=neighbor(i,kk);
      if (distance(i,k) >= cutoff) continue;
      threebody_center(i,j,k,fi);
      force[i] += fi;
    }

    num_k_nbors=get_neighbor_count(j);
    for (kk=0; kk<num_k_nbors; kk++) {
      k=neighbor(j,kk);
      if (k == j || k == i) continue;
      if (distance(i,k) >= cutoff) continue;
      threebody_end(i,j,k,fi);
      force[i] += fi;
    }
  }
```

In this case, an additional inner loop is added and the force computation is divided into two separate kernels. The first inner loop is similar, and handles the case where the j and k atom in the triplet are both within a distance less than r_α, the sum of the potential cutoff and the neighbor list skin [18]. For the single triplet illustrated in Figure 2 this loop updates the force when i equals atom 2. The loop is changed such that only the force for atom i is updated. Therefore, the force kernel can potentially be simplified to only include terms relevant to atom i (this depends on the model used). When i is atom 1 or 3 in Figure 2, the force must also be updated for the triplet, but in this case, the j and k atoms are not necessarily within the cutoff. This can be handled by either increasing the skin such that it is greater than the cutoff or by looping over neighbors of atom j rather than i in the inner loop. The latter approach of looping over neighbors of neighbors (used in the listing), will almost always be more efficient. Again, a separate kernel with reduced computation can potentially be used to only calculate terms for the triplet necessary for the i atom.

When implementing the RCA for multiple accelerators with distributed memory, additional issues arise. For a spatial decomposition in MD, each process stores *ghost* atoms at the borders of the subdomain that can be within a distance r_α of the *local* atoms simulated on the process (Figure 2). For the NA, atoms within the borders have forces with contributions calculated on two different processes (because only a single neighbor list is used for each triplet). Therefore, communication is required at every timestep to accumulate forces for the ghost atoms. For the RCA, however, neighbors of neighbors are used in the force computation. Therefore, neighbor lists are required not only for local atoms, but also for ghost atoms when using the RCA. In order to avoid including the force contribution for a triplet more than once, the neighbor lists for ghosts must either distinguish local and ghost neighbors or the length of the border must be doubled to allow the forces to be calculated entirely on a single MPI process. The latter increases the amount of communications for the ghost-atom data exchange, but eliminates the need for communication in force accumulation. This is the approach that we use here.

Depending on the study, calculation of energy and virial terms might be necessary on some or all timesteps. If only global energies and pressures are needed, they can be calculated entirely in the threebody_center kernel in a manner similar to the NA. If per-atom energies or stresses are required, the calculation must occur in both 3-body kernels. In this case, the kernels will probably have to perform the full 3-body calculation that is not simplified for the redundant computation. The RCA requires up to 3X the number of calculations when compared to the NA, requires calculation of neighbor lists for ghost atoms, and an increase in the number of ghost atoms with a doubling of the length of the borders. To offset this increase, however, the number of force updates in global memory is reduced from $n + n \cdot \frac{b_n^2+b_n}{2}$ to n and the requirement for atomic operations is eliminated. Additionally, for the RCA described here, the requirement for interprocess communication for force accumulation is eliminated.

For the RCA, the nested force computation loop must be parallelized across the work-items. For an atom decomposition, this is done by parallelizing the i loop to assign an atom to each work-item, with the work-item performing the entire force accumulation for that atom. This approach can lead to significant performance gains, however, it is not ideal for GPU accelerators. The first issue is work-item divergence where cores are effectively idle when j atoms are not within the cutoff. The second issue is that this limits the number of work-items that can be used for a given number of atoms. Accelerators and many-core chips require a large number of work-items for efficient performance when compared to traditional CPUs and for many it is desirable to have many more work-items than cores in order to hide latencies. This is an important concern for HPC implementations where it is often desirable to scale up to large node counts, reducing the number of atoms per MPI process to decrease the time to solution. Although for large atom counts a difference is not observed, we have shown that force decompositions can provide substantial improvements in parallel effi-

thread	1	2	3	4	5	...
atom	i	i	i	i	i+1	...
nbor 1	jj	jj	jj+1	jj+1	jj	...
nbor 2	kk	kk+1	kk	kk+1	kk	...

Figure 3: Illustration showing the atom and neighbor indices for the static regular force decomposition.

ciency for pairwise models [19].

Force decompositions also divide the inner loops among work-items. This increases the number of work-items available to keep cores busy and also amortizes the access latencies. The tradeoff is an increase in the amount of computation required for additional reductions and in some cases, the requirement for additional work-item synchronizations [19]. We tested many different combinations for parallelizing the jj and kk loops between an arbitrary number of work-items. Work-item divergence occurring at the cutoff check in the jj loop is a significant concern, because this divergence occurs for the duration of the inner kk loop. Therefore, we tested parallelization where the jj loop was split across a number of work-items equal to the number of cores that are restricted to perform the same instruction. This eliminates divergence at the jj loop cutoff check, but because this number is high for the Nvidia hardware used (32), the overhead for the reduction was too large at typical neighbor list sizes. In our tests, we observed that the optimal number of work-items for parallelization of both the jj and kk loops was determined by the neighbor list size. Based on these results, we implemented a static regular force decomposition (SRFD) where a single parameter is used to determine the number of work-items for the parallelization; the parameter is constant for all atoms throughout the duration of the simulation. Although this works well for Nvidia hardware, the advantages of the approach might be vendor dependent. We have left this for a future study.

In the SRFD, a single work-item parameter, w_n is specified. The optimal choice for w_n depends on the neighbor list size and the hardware, however, defaults are chosen at runtime in LAMMPS based on the typical neighbor list size and the detected hardware. The force accumulation for each atom is performed by w_n^2 work-items with w_n work-items performing the innermost (kk) loop for each neighbor, jj (Figure 3). The neighbors are stored in memory such that w_n neighbors for an atom are contiguous in memory and such that the neighbors for i and $i+1$ are contiguous. In the case where $w_n = 1$, the SRFD is equivalent to an atom decomposition. The condition that $kk > jj$ for the innermost loop in the RCA listing will lead to an imbalance in the amount of computation assigned to each work-item. This can potentially be addressed by looping from 0 to $jnum - 1$ and only computing the force when $mod(jj + kk, 2) = 0$ when $jj > kk$ or $mod(jj + kk, 1) = 0$ when $jj < kk$ [27]. This approach can balance the amount of computation for each work-item if implemented in a manner that does not exacerbate work-item divergence. The SRFD approach decreases the impact from work-item divergence in our tests and allows for substantially better performance at smaller atom counts.

2.5. Accelerating Stillinger-Weber

For this paper, we have chosen to evaluate acceleration of 3-body interactions using the RCA with SRFD parallelism for the Stillinger-Weber potential. The Stillinger-Weber model was first introduced in 1985 to probe the behaviors of four-bonded monoatomic systems such as carbon, silicon and germanium [15]. More recently, it has been parameterized to represent water molecules, which although multi-atomic, share many fundamental properties with elemental silicon and germanium. The tunable parameters of mW water have been optimized to yield tetrahedral ordering in between that of carbon and silicon; this lead to comparable or better accuracy than the most popular point-charge models at orders-of-magnitude faster wall-clock time [20].

The Stillinger-Weber potential is given by,

$$U = \sum_i \sum_{j>i} \phi_2(r_{ij}) + \sum_i \sum_{j\neq i} \sum_{k>j} \phi_3(r_{ij}, r_{ik}, \theta_{jik}) \quad (2)$$

$$\phi_2(r_{ij}) = A_{ij}\epsilon_{ij}\left[B_{ij}(\frac{\sigma_{ij}}{r_{ij}})^{p_{ij}} - (\frac{\sigma_{ij}}{r_{ij}})^{q_{ij}}\right] \cdot \exp\left(\frac{\sigma_{ij}}{r_{ij} - a_{ij}\sigma_{ij}}\right) \quad (3)$$

$$\phi_3(r_{ij}, r_{ik}, \theta_{jik}) = \lambda_{jik}\epsilon_{jik}\left[\cos\theta_{jik} - \cos\theta_{0jik}\right]^2 \cdot \exp\left(\frac{\gamma_{ij}\sigma_{ij}}{r_{ij} - a_{ij}\sigma_{ij}}\right)\exp\left(\frac{\gamma_{ik}\sigma_{ik}}{r_{ik} - a_{ik}\sigma_{ik}}\right), \quad (4)$$

where i indexes the center atom, θ_{jik} the angle between the atoms, and r_{ij} and r_{ik} give the interatomic separations. The other letters denote empirical parameters for the element types based on the model. The Stillinger-Weber potential consists of a 2-body term, ϕ_2, and a 3-body term, ϕ_3. In order to provide a balanced computational workload to work-items for the Stillinger-Weber model, three kernels are used. This includes a kernel for the 2-body term, implemented similar to other accelerated pairwise potentials in LAMMPS [18], and two kernels for the 3-body terms and described above for the RCA. Because there are no data dependencies for these kernels, they can be calculated independently and concurrently. The code can be compiled for concurrent force calculation (CFC) in which case the 2-body and threebody_center kernels are launched in a separate command queue from the threebody_end kernel in order to allow for concurrent execution. This is potentially advantageous for small atom counts in order to increase the number of work-items in flight at any given time. Because this requires an additional synchronization and reduction, CFC is optional and for testing purposes.

A default w_n parameter for the SRFD is chosen at runtime based on the hardware or the user can optionally specify this parameter. The w_n parameter determines the parallelization of the RCA as described above and also the number of work-items

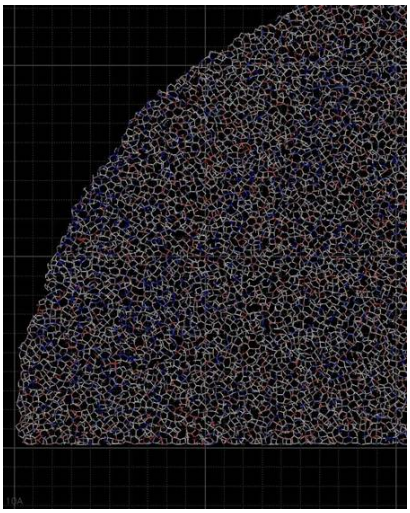

Figure 4: Close-up view of million molecule mW water droplet on tunable contact-angle surface. Lines indicate bonds among the mW water molecules. White denotes average particle mobility, red denotes higher-than-average particle mobility and blue denotes lower-than-average particle mobility. Rendered with VMD [29].

that perform the force accumulation for the 2-body term as described previously [19]. Neighbor list calculation is performed on the device as described previously [18] with the exception that neighbor lists are also calculated for ghost atoms. As discussed above, interprocess communication for ghost-atom data exchange is removed from the accelerated version in trade for a border length for ghost-atom data exchange that is double that in the CPU-only version.

2.6. Benchmarks

We have benchmarked the strong and weak scaling performance of bulk water in a cubic box with periodic boundary conditions, comparing the performance of mW water against that of one of the least expensive point charge models, SPC/E [30]. We have also benchmarked the performance of a production run comprising a water droplet with one million mW molecules placed on a tunable contact-angle surface, where the walls and ceiling of the simulation box are reflecting (Figure 4) [31]. For the droplet, we model one million molecules, which is orders of magnitude larger than typical simulations, to minimize any finite-size effects that could lead to the erroneous interpretation of simulation results. In particular, given our desire to probe spontaneous nucleation behaviors in droplets, our system size must be at least several times larger than the critical nucleus size. Moreover, the surface-area-to-bulk ratio must be sufficiently small that surface effects not unduly dominate.

All of the simulations were conducted using the canonical ensemble where the relaxation time of the Nose-Hoover thermostat was 1 ps. For mW, the time step was 10 fs and for SPC/E, it was 1 fs. A neighbor list skin of 1Å was used. For mW, neighbor list builds were forced to be at least 2 timesteps apart, with checking every 2 timesteps for atom movement requiring a new build. For SPCE/E, the builds were forced to be at least 6 timesteps apart. The LAMMPS "grid numa" option was used to optimize MPI process mapping to reduce off-node communications [10]. A warmup run of 20 timesteps was used followed by a 400 timestep run used for the timings presented here. Single precision FFTs were used for long-range electrostatics to reduce MPI communications. All benchmarks using the GPUs on Titan were performed with mixed precision as opposed to full double precision for the CPU-only runs [18].

2.7. Titan XK7 Supercomputer

For the benchmark simulations performed here, we used the Titan supercomputer at Oak Ridge National Laboratory. Titan is a Cray XK7 computer with 18688 compute nodes, 512 XIO nodes, and a Gemini interconnect. Each node holds a single 16-core AMD Opteron 6274 running at 2.2 GHz with 32GB ECC DDR3 SDRAM. The Opteron is connected to a Tesla K20X via PCI-e 2.0. The K20X has 2688 compute cores running at 732 MHz with 6GB of GDDR5 SDRAM. The Gemini interconnect is connected in a 3-D torus topology and has 1-2 microsecond latency for point-to-point messages and 20 GB/s of injection bandwidth per node. At the time of the benchmarks, Titan was running version $4.1u2$ of the Cray Linux Environment compiled with version 304.47.13 of the Nvidia CUDA driver. The 2013 April 24th version of LAMMPS was used with the modifications described below. The code was compiled using version 4.1.40 of the Cray GNU programming environment with gcc version 4.7.2 and nvcc version 5.0. The Nvidia proxy server was used for runs with more than one MPI process sharing the device in order to allow for context sharing and concurrent execution/data transfer from different processes. Runs using less than 9 MPI processes per node were launched with the core affinity set so that only 1 core per AMD Bulldozer module was used.

3. Results

Figure 1 shows the timing results for simulation of a 32,000 molecule water box with LAMMPS when run on the CPU of a single XK7 node. The simulation uses the Stillinger-Weber potential with the mW parameterization. In this case, 16 cores are used and the entire simulation time is 11.4 seconds for 400 timesteps. The force calculation is 89.6% of the simulation time with 6.2% for neighbor list builds, 3.65% for MPI communications, and 0.55% for time integration and other statistics calculations. Most of the force calculation time, 95%, is for calculation of 3-body interactions.

The results with acceleration using the RCA are shown in Figure 5. For a single MPI process using an atom decomposition ($w_n = 1$), the simulation time is reduced to 3.66 seconds with 1.47 seconds for the force computation and 0.89 seconds for the neighbor list build. Most of the time for force computation, 96.8%, is used for calculation of 3-body interactions. Although we are performing the neighbor list build on the accelerator, the time required is 26% slower than neighbor list calculation on 16 cores using the CPU. In part this is due to the increase in the number of atoms requiring neighbor list builds since lists are also required for ghost atoms. Performing

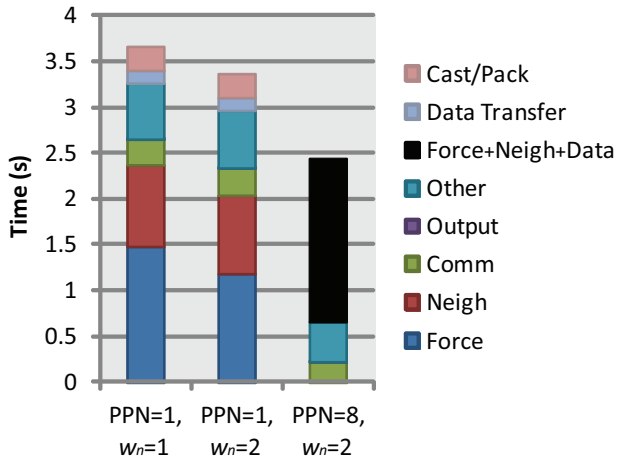

Figure 5: Time for simulation of a 32000 molecule water box with mW on a single XK7 node with acceleration. PPN is the number of MPI processes sharing the accelerator. $w_n = 1$ is an atom decomposition and $w_n = 2$ is a force decomposition with 4 threads assigned to each atom. The profile shows time for calculation of forces, energies, and virials (Force), neighbor-list builds (Neigh), MPI communications or memory copies (Comm), screen/file output (Output), time integration, statistics and other calculations (Other), host-device data transfer (Data Transfer), and time on the host for casting and packing data for transfer (Cast/Pack). "Force+Neigh+Data" combines "Force", "Neigh", "Data Transfer", and "Cast/Pack" when individual timings are not available. Here, acceleration is for "Force" and "Neigh" calculations.

the neighbor build on the accelerator reduces substantially the amount of host-device data transfer required, however, since the neighbor list storage typically dominates the memory usage for molecular dynamics. With neighbor list builds on the GPU, the host-device data transfer represents 4.1% of the total simulation time.

Using a force decomposition with $w_n = 2$ reduces the time for force calculation by 20%. Although there is additional code required for storage of neighbors to allow for contiguous memory access with the SRFD, the neighbor list build time is not impacted. The choice of $w_n = 2$ will depend on the device, the cutoff, and the number of particles on the accelerator. Devices with more cores and devices that benefit from oversubscribing the cores will favor larger values of w_n. Models with larger cutoffs see much more substantial benefits with higher values of w_n. For the model and devices used here, $w_n = 2$ performed best at larger molecule counts with a 12% reduction in force time with 256000 molecules and a 40.6% with 8000 molecules. For 4000 and 2000 molecules on the device, $w_n = 4$ performed best with up to a 78.6% reduction in force calculation time.

Running with a single MPI process using the device has the advantage that there is no overhead from scheduling or handling driver requests from multiple processes. Additionally, in the single node case in Figure 5, MPI communications are replaced with memory copies. (Because LAMMPS is intended as a parallel code, atoms across periodic boundaries are treated as ghost atoms, even when using a single MPI task. Therefore, the packing of atom data at borders for communication still occurs, although memory copies are used in place of the MPI calls). When using a single process however, routines on the CPU, such as time integration, thermostats, barostats, and other statistics, are performed in serial. In addition to parallelizing these routines, running multiple MPI processes sharing the accelerator has the advantage of allowing pipelining for host-device data transfers and kernel execution. That is, the device can overlap host-device communications from one process with force calculations from another. Due to these benefits, we see a significant performance improvement when sharing the device between multiple MPI processes. For the single node 32000 molecule water box simulation in Figure 5, there is a 27.5% reduction in overall simulation time when using 8 MPI processes. In this case, using the accelerators on the XK7 results in a speedup of 4.68. 17.8% of the simulation time is spent in various CPU calculations with another 9% required for interprocess communication. We note that there is a slight decrease in the time for MPI communications versus the single process memory copies and data packing. Because the amount of data per MPI process for exchange of ghost-atom forces scales sublinearly for a fixed-size simulation, the communication and data-packing time can be reduced when using multiple processes running on a single node (this, of course, will depend on the hardware and, in the case of multiple nodes, can result in slower times despite the reduction in data).

We also tested LAMMPS compiled to allow concurrent force calculation (CFC) in which case the device can choose to run multiple force calculation kernels concurrently on the device. Although we did see some performance improvement for smaller molecule counts with a single MPI process, in most cases there was a noticeable performance degradation from the additional synchronization and reduction overhead. The approach was never faster when using multiple MPI processes sharing the device (in which case kernels from different processes can run concurrently) and therefore we have not enabled this option in LAMMPS.

Results from parallel simulations using multiple GPUs are shown in Figure 6. In the strong scaling tests, a fixed simulation size of 256000 molecules is benchmarked using between 1 and 128 nodes. The strong scaling benchmarks test the ability to reduce the time to solution for a given simulation. For the mW model on a single node, the simulation rate is 5.44 times faster when using acceleration. On 128 nodes, this speedup is reduced to 1.48 with a simulation rate slightly over 0.5 μs per day. This reduction in relative performance on the GPU device is expected at lower molecule counts; more work is required on each node in order to effectively utilize the thousands of cores on each accelerator. In the weak scaling tests, the number of molecules per node is held constant at 32000; this benchmarks the ability to run larger simulations on more nodes. In this case, the simulations with acceleration are 4.71 times faster on a single node and 3.52 times faster on 1024 nodes resulting in a parallel efficiency of 74.7%.

The mW model is intended to allow faster simulations of water with coarse-grain simulations that reduce the number of particles and allow for larger timesteps. Additionally, long-range electrostatic terms are not included. In order to maintain accuracy, a 3-body potential is used. Therefore, we also compared performance to simulation with the SPC/E water model. For

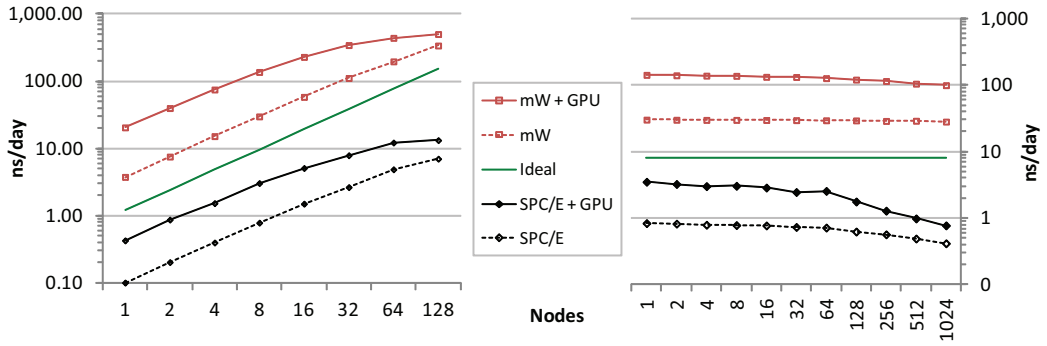

Figure 6: Simulation rates for a fixed-size 256K molecule water box in parallel (Left) and a scaled-size water box with 32K molecules per node (Right). Parallel simulation rates with ideal efficiency would have a slope equal to that of the green line in each plot.

Table 2: Summary of best speedups versus running on a single XK7 CPU for CPU-only and accelerated runs. Simulation is 400 timesteps for a 1 million molecule droplet. The speedups are calculated based on the single node loop time of 440.3 seconds.

	1 node		64 nodes	
Test case	Cores	Speedup	Cores	Speedup
XK7 w/out GPU	16	1.0	1024	41.6
XK7 w/ GPU	16	6.6	512	211.0

the strong scaling, GPU acceleration for the SPC/E model improves performance on a single node by 4.24X and by 1.89X on 128 nodes. Although the short-range force calculation in SPC/E is a smaller fraction of the simulation time, GPU acceleration still provides significant improvements because of efficient overlap of short-range, bond, and long-range forces with concurrent CPU and accelerator calculations [19]. Additionally, the number of particles is higher in the SPC/E case, with 3 times the number atoms for the same number of water molecules. For weak scaling, however, the parallel efficiency is impacted by the effectively all-to-all communications for the FFT-based Poisson solve. The speedup on 1 node is 4.18 versus 1.86 on 1024 nodes. Although there are methods to reduce the impact of the long-range electrostatics solve (such as multiple timestepping), the mW model offers speedups of over two orders of magnitude due to the coarse-grain model employed and the absence of long-range electrostatics.

Specifically, the simulation loop time on a single XK7 node with 256K molecules is 4.8 times faster when using the mW model instead of SPC/E; on 128 nodes it is 3.8 times faster. However, because the coarse-grain mW model allows a timestep to be used that is 10 times larger, the actual simulation rates are 48 and 38 times faster respectively. Because the mW model does not consider long-range electrostatics, the gains from use of the mW model become more pronounced with larger simulations on more nodes. In the weak scaling tests, the simulation rate on a single node is 40.5 times faster; on 1024 nodes it is 127.8 times faster.

For the final evaluation of mW performance, we compared simulation of a water droplet on a substrate with LAMMPS scripts used for production simulations. The simulations consist of a 1 million molecule water droplet interacting with a 9-3 Lennard-Jones wall for the substrate. The results are summarized in Table 2. On a single XK7 node, we are able to achieve simulation rates that are 6.6 times faster when using acceleration with the RCA. On 64 nodes, the simulations are 5.1 times faster. The relative performance in this case is slightly better than for a bulk water box with a similar molecule count. This is due to the fact that a perfectly balanced spatial decomposition to divide the molecules between processes cannot be achieved in LAMMPS (which requires rectangular subdomains for dynamic load balancing) and also because the communications topology for the simulations is different. For the latter, the simulation box is factored such that each process only has four neighbors and there is no division of work in the dimension normal to the substrate.

4. Discussion

Potential energy models with 3-body interactions are essential to many studies using molecular dynamics. The development of algorithms to run these simulations on hybrid machines with coprocessors or accelerators is critical to achieving performance gains on current and future HPC systems. Because of the model complexities and additional data dependencies, implementation of 3-body interactions requires more substantial modifications to traditional MD codes. We have shown, however, that a conceptually simple approach can be used to reduce dramatically global memory access for these models while eliminating data dependencies to allow for a deterministic code. Although this approach significantly increases the number of floating point operations required to perform force calculations, we have shown that substantial performance gains can still be realized because of the inherent parallelism in the approach and the fact that the 3-body interaction typically dominates the simulation time. This approach can be used for efficient acceleration of many important potentials including MEAM, Tersoff, REBO, AIREBO, Stillinger-Weber, Bond-Order Potentials, and others.

Although porting existing models for hybrid machines is important, if the current trend in computing continues, it will be necessary to also consider the development of new models that allow for better data locality with computations that can better exploit massive concurrency [32]. That is, because clock speeds are no longer increasing and data movement is becoming the dominant bottleneck, we cannot expect to continue to get increased performance on future machines that add more cores without reconsidering the models that are used. Therefore, research into novel approaches for simulation that trade increased computation for greater accuracy and simulation rates are critical. For the mW model used here, for example, researchers can achieve simulation rates that are orders of magnitude higher despite the additional complexity in the model.

Our work for acceleration of Stillinger-Weber can have an immediate impact on MD employing the mW model. There is great interest in understanding the microscopic mechanism of droplets freezing on surfaces, and yet probing this behavior experimentally is extremely challenging. Molecular dynamics is a tool that allows such molecular-level interactions to be probed, but the requirement for large system sizes and in particular very long simulation times renders modeling cost-prohibitive in most simulations of rare, activated processes. By accelerating the simulation time, the barrier to adoption is greatly reduced, opening up the possibility of using molecular dynamics as a valuable tool that complements experimental findings.

5. Acknowledgements

This research was conducted in part under the auspices of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC. This research was also conducted in part under the auspices of the GE Global Research High Performance Computing program. This research used resources of the Leadership Computing Facility at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes. All of the code described in this paper is available in the open-source LAMMPS software package, available at http://lammps.sandia.gov/ or by contacting the authors.

References

[1] B. Bland, in: High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:, IEEE, pp. 2189–2211.
[2] Top500, Top500 Supercomputer Sites http://www.top500.org, (accessed May 6, 2013).
[3] Green500, The Green 500 http://www.green500.org, (accessed May 6, 2013).
[4] H. J. D. Baker, J. A., Molecular Informatics 30 (2011) 498–504.
[5] M. Harvey, G. De Fabritiis, Wiley Interdisciplinary Reviews: Computational Molecular Science 2 (2012) 734–742.
[6] S. J. Plimpton, A. P. Thompson, MRS Bull 37 (2012) 513–521.
[7] J. E. Stone, D. J. Hardy, I. S. Ufimtsev, K. Schulten, Journal of Molecular Graphics and Modeling In Press (2010).
[8] M. S. Daw, M. Baskes, Physical Review Letters 50 (1983) 1285–1288.
[9] I. Morozov, A. Kazennov, R. Bystryi, G. Norman, V. Pisarev, V. Stegailov, Computer Physics Communications 182 (2011) 1974–1978.
[10] W. M. Brown, T. D. Nguyen, M. Fuentes-Cabrera, J. D. Fowlkes, P. D. Rack, M. Berger, A. S. Bland, Procedia Comput. Sci. 9 (2012) 186–195.
[11] M. Baskes, Physical review letters 59 (1987) 2666–2669.
[12] J. Tersoff, Physical Review B 37 (1988) 6991.
[13] D. W. Brenner, Physical Review B 42 (1990) 9458.
[14] S. J. Stuart, A. B. Tutein, J. A. Harrison, The Journal of Chemical Physics 112 (2000) 6472.
[15] F. H. Stillinger, T. A. Weber, Physical Review B 31 (1985) 5262.
[16] D. Pettifor, I. Oleinik, Physical Review B 59 (1999) 8487.
[17] C. Hou, J. Xu, P. Wang, W. Huang, X. Wang, Computer Physics Communications (2013).
[18] W. M. Brown, P. Wang, S. J. Plimpton, A. N. Tharrington, Computer Physics Communications 182 (2011) 898–911.
[19] W. M. Brown, A. Kohlmeyer, S. J. Plimpton, A. N. Tharrington, Comp. Phys. Comm. 183 (2012) 449–459.
[20] V. Molinero, E. B. Moore, The Journal of Physical Chemistry B 113 (2008) 4008–4016.
[21] E. B. Moore, V. Molinero, Nature 479 (2011) 506–508.
[22] M. Matsumoto, S. Saito, I. Ohmine, Nature 416 (2002) 409–413.
[23] M. Yamada, S. Mossa, H. E. Stanley, F. Sciortino, Physical review letters 88 (2002) 195701.
[24] D. W. Oxtoby, Journal of Physics: Condensed Matter 4 (1992) 7627.
[25] A. Alizadeh, M. Yamada, R. Li, W. Shang, S. Otta, S. Zhong, L. Ge, A. Dhinojwala, K. R. Conway, V. Bahadur, et al., Langmuir 28 (2012) 3180–3186.
[26] A. Alizadeh, V. Bahadur, A. Kulkarni, M. Yamada, J. A. Ruud, MRS Bulletin 38 (2013) 407–411.
[27] S. Plimpton, Journal of Computational Physics 117 (1995) 1–19.
[28] J. Sumanth, D. R. Swanson, H. Jiang, in: Proceedings of the 21st annual international conference on Supercomputing, ACM, pp. 105–115.
[29] W. Humphrey, A. Dalke, K. Schulten, Journal of molecular graphics 14 (1996) 33–38.
[30] H. Berendsen, J. Grigera, T. Straatsma, Journal of Physical Chemistry 91 (1987) 6269–6271.
[31] M. Yamada, A. Alizadeh, B. J. Moore, Submitted (2013).
[32] T. D. Nguyen, J.-M. Y. Carrillo, A. V. Dobrynin, W. M. Brown, Journal of Chemical Theory and Computation 9 (2012) 73–83.